# Lab3 Cache设计指导

Cache实验分为两个阶段，这里是cache实验的第一阶段指导文档。在本阶段，需要完成Cache 的实现和独立测试。在第二阶段，将进行Cache 与Lab2的四级流水线CPU组合设计，并对benchmark性能进行测试，具体请见文档《Lab3-Cache实验指导.docx》。

## 实验目的

1.权衡cache size增大带来的命中率提升收益和存储资源电路面积的开销

2.权衡选择合适的组相连度（相连度增大cache size也会增大，但是冲突miss会减低）

3.体会使用复杂电路实现复杂替换策略带来的收益和简单替换策略的优势（有时候简单策略比复杂策略效果不差很多甚至可能更好）

4.理解写回法的优劣

## 实验要求

**阶段1：**

理解我们提供的直接映射策略的cache，将它修改为N路组相连的cache，并通过我们提供的cache读写测试。

我们提供：

1. 一个简单的直接映射、写回带写分配的cache代码(cache.sv)
2. 一个python脚本(generate\_cache\_tb.py)，该脚本用于生成不同规模的testbench文件。这些testbench文件对cache进行多次随机读写，最后从cache中读取数据并验证cache的正确性。同时，我们也提供一个已经生成的testbench。
3. 一份文档《cache设计指导.docx》，解读我们提供的cache的结构和时序，并简单的介绍N路组相连cache的修改建议。

要求：阅读并理解我们提供的简单cache的代码，将它修改为N路组相连的（要求组相连度使用宏定义可调）、写回并带写分配的cache。要求实现FIFO、LRU两种替换策略。

使用提供的python脚本生成一个新的testbench，并对自己的cache进行验证（要求FIFO和LRU策略都要验证，并修改组相连度等参数进行多次验证），在实验报告中解释自己所编写的代码。

**阶段2：**

使用阶段一编写的N路组相连cache，正确运行我们提供的几个程序。

要求在cpu中加入你所编写的N路组相连的cache，并要求能成功运行这个排序算法（所谓成功运行，是指运行后的结果符合预期）

## 实验报告

* 实验目标
* 实验环境和工具
* 实验内容和过程（使用我们提供的快速排序和矩阵乘法的benchmark进行实验，鼓励自己编写更多的汇编benchmark进行测试，体会cache size、组相连度、替换策略针对不同程序的优化效果，以及策略改变带来的电路面积的变化。针对不同程序，权衡性能和电路面积给出一个较优的cache参数和策略。其中“性能”参数使用运行仿真时的时钟周期数量进行评估。“资源占用”参数使用vivado或其它综合工具给出的综合报告进行评估。进行这一步时需要用阶段二的结果进行一些实验，不能仅仅进行理论分析，实验报告中需要给出实验结果（例如仿真波形的截图、vivado综合报告等）。）
* 实验总结（说说自己踩的坑，总结收获，分析下自己花了多少时间，都用来做什么事情）
* 提出改进实验的意见

**实验总结是每位同学写自己的，其他部分同一个小组写一份，合并在一个报告里面即可，最后整个小组上交一份报告，报告最后写上分数分配。**

1. 暂时只做dcache，icache默认不缺失，仍然使用原有代码充当icache

2. 在进行cache实验时，为了方便Verilog编写，一律不需要处理读写的独热码，只需考虑sw和lw这两种“整字读写“的指令。我们提供的相关benchmark中的所有load、store指令也将只有sw和lw。

3. 我们的代码利用封装的Bram模拟DDR，Cache命中时间为1 cycle，模拟DDR命中时间设置为50 cycle（因为真实情况下 cache命中时间为1ns，DDR为50-100ns）

## Cache 实验常见疑问：

**问：无论进行快速排序，还是矩阵乘法，最终的主存（mem.sv 模块里的ram\_cell变量）里的数据都与正确结果整体上是相同的，但会略有差异，是为什么？**

**答：因为是写回策略的cache，所以最终会有一些数据还在cache中未写入主存。属于正常现象。但如果你的cache写错了，那么快速排序和矩阵乘法的结果就会很离谱。在检查实验时，助教主要通过第一阶段cache\_tb.sv （即脱离CPU的cache检验）去判断你的cache的正确性。**

**问：Cache实验都检查什么内容**

**答：阶段1的cache\_tb必须正确通过，然后需要向助教展示你所写的cache代码，讲解你的思路，包括但不限于如何在原有基础上加并行TAG比较、如何实现LRU和FIFO换出策略。**

**阶段2的分值不高，只需要展示你所写的LRU和FIFO策略的cache能够跑通快速排序和矩阵乘法就行，对于快速排序来说，跑通的结果应该是主存中的数据变成有序的。对于矩阵乘法来说，跑通的结果应该是主存中的数据与注释中的数据相同。当然，因为有些数据在cache中还未写入主存，所以允许有一些不同。**

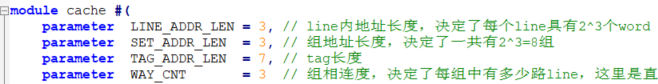
**问：如何写cache实验的报告**

**Cache实验报告在cache实验中占很大的分值比重。因为Cache实验是相对于CPU流水线设计而言是开放的，重点在权衡性能和面积的体会，而Cache本身的实现只是前提。阶段二所提供的快速排序和矩阵乘法的benchmark就是用来在实验报告中进行实验分析的，也鼓励自己编写更多的汇编benchmark进行测试。在做实验时，修改Cache的参数，体会cache size、组相连度、替换策略针对不同程序的优化效果，以及策略改变带来的电路面积的变化。针对不同程序，权衡性能和电路面积给出一个较优的cache参数和策略。注意，需要权衡性能和电路面积。**

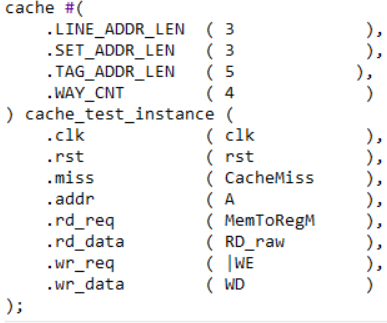
**其中“性能”参数使用运行仿真时的时钟周期数量进行评估。“资源占用”参数使用vivado给出的综合报告进行评估。在写报告时，不能仅仅进行理论分析，实验报告中需要给出实验结果（例如仿真波形的截图、vivado综合报告等）。**

**问：如何修改cache的参数**

**答：cache.sv的开头中有如下的parameter定义：**

****

**这些参数应该在编写实验报告时进行修改，用于权衡性能和电路面积。修改方法是在调用它的模块（即WBSegReg.v）中，修改这几个参数，如下：**

****

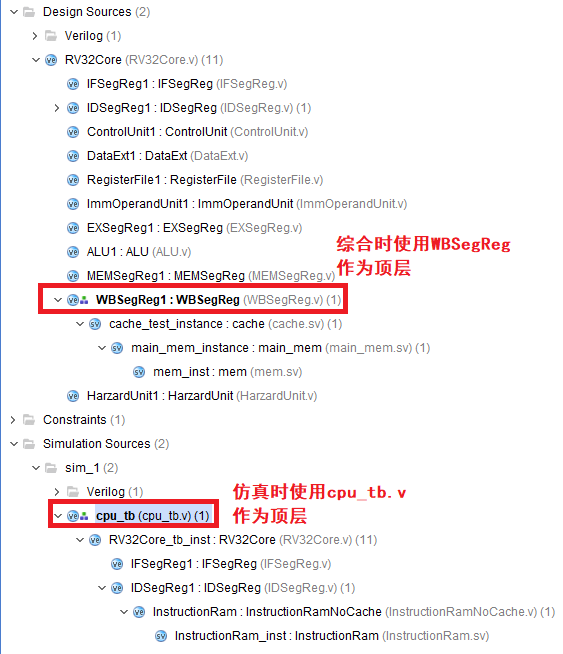
**Verilog/SystemVerilog 中的parameter类似C++中的默认参数，在被调模块中只是指定默认值。在主调模块中，如果指定，则使用该指定值；如果不指定，则使用默认值。**

**问：vivado的综合和仿真是否互相依赖？**

**答：没有，综合前不需要仿真，仿真前也不需要综合。**

**问：为了进行电路面积评估，如何进行综合？**

**答：因为我们只关心cache的电路面积，而CPU我们是不会修改的，所以建议在综合前，将WBSegReg.v 设置为顶层，可以提高综合速度。在vivado中应该如下设置仿真和综合的顶层：**

****

**因为仿真与综合互相独立，所以可以使用不同的顶层。设置cpu\_tb.v作为仿真的顶层，用于进行cache性能的评估。设置WBSegReg.v作为综合的顶层，综合后查看综合报告，用于进行电路面积的评估。**